AMO - Advanced Modeling and Optimization, Volume 5, Number 1, 2003

ON THE GENERATION OF P-SEQUENCES *

H. AHRABIAN
Department of Mathematics and Computer Science,
Faculty of Science, University of Tehran,
Tehran, Iran.
ahrabian@khayam.ut.ac.ir

and

A. NOWZARI-DALINI
Department of Mathematics and Computer Science,
Faculty of Science, University of Tehran,
Tehran, Iran.
nowzari@khayam.ut.ac.ir

Abstract

An efficient algorithm for the generation of P-sequences is presented. P-sequences are integer sequences characterizing all shapes of n-noded binary trees. This algorithm generates each sequence in B-order with constant average time $O(1)$. The sequences are generated in lexicographical orders. The ranking and unranking algorithms with $O(n)$ time complexity are also described. Finally, an algorithm for the construction of a binary tree with a linked structure from P-sequence is presented.

Keywords: Binary tree, Recursion, P-sequences, B-order.

*This research is supported by University of Tehran.

1. Introduction

Binary trees are of fundamental importance in computer science and one the most basic and simple data structures. It can be used to maintain any ordered set that must be accessed and updated. The problem of coding binary trees has received much attention because of its practical relevance. There have been many coding schemes proposed in the literature [Ahrabian and Nowzari, 1999; Bultena and Ruskey, 1998; Gupta, 1991; Pallo and Racca, 1985; Ruskey and Hu, 1977; Vajnovszki, 1998; Zaks, 1980]. In some of these papers binary trees are represented by integer sequences and the corresponding sequences are generated [Ahrabian and Nowzari, 1999; Gupta, 1991; Pallo and Racca, 1985; Vajnovszki, 1998; Zaks, 1980].

P-sequences introduced by Pallo and Racca [1985] are integer sequences characterizing all shapes of n-noded binary trees. The P-sequence of a binary tree T is an integer sequence $P_T = (p_1, p_2, \cdots, p_{|T|})$, where p_i is the number of visited internal nodes before the ith external node in the preorder traversal of T, and $|T|$ denotes the number of internal nodes in the binary tree, it is noted that the assigned value to the $|T| + 1$ external node is always equal to $p_{|T|} = n$ and therefore it is discarded. We present here an efficient algorithm that generates each sequence in constant average time $O(1)$. Our algorithm generates the corresponding binary trees in B-order. Recall from Pallo and Racca [1985], given two trees T and T' with n nodes, $T < T'$ are in B-order if P_T is lexicographically less that $P_{T'}$ (for the other B-order generation algorithms see [Ruskey and Hu, 1977; Zaks, 1980]).

The previous algorithms presented by Pallo and Racca [1985] and Gupta [1991] also generate the P-sequences in B-order and Ballot-order respectively in constant average time $O(1)$. It should be noted that Ballot-order is an order for the codes not for binary trees and this order is the same as Ballot-sequence order [Rotem, 1975; Rotem and Varol, 1978]. Our algorithm can be modified to generates the P-sequences in Ballot-order.

The ranking and unranking algorithms with $O(n)$ time complexity are also presented. The time complexity of previous ranking and unranking algorithms presented in [Pallo and Racca, 1985] is $O(n^2)$. In addition to the generation algorithm, an algorithms for the construction of a binary tree with a linked structure from a P-sequence is discussed.

2. The Generation Algorithm

In this section we describe an algorithm for the generation of P-sequences. Clearly, for the given binary tree T denoted in Figure 1, the P-sequence of T is the integer sequence $P_T = (3, 3, 3, 5, 5, 7, 7)$, where each integer in position i shows the number of visited internal nodes before the ith external node in a preorder traversal.

The algorithm *GenP-Seq* given in Figure 2, generates P-sequences in the reverse order of the B-order. This algorithm is recursive and has four parameters: $'P'$, $'k'$, $'l'$ and $'q'$, where $'P'$ is an integer array of size n and initially is equal to $(n, n, \cdots, n)$. The parameter $'k'$ is used for constructing the sequence and initially is equal to n. The two parameters $'l'$ and $'q'$ control the number of recallings of the algorithm from the two underlying recursions in the algorithm. The initial value of $'l'$ is equal to $n-1$, and $'q'$ is equal to 1. The algorithm produces each code by decrementing the elements of $'P'$ by 1, from the leftmost element, one by one. The generation starts from $P = (n, n, \cdots, n)$ and all the elements of $'P'$ become $n-1$ except the nth element, then the decrementation restarts from the beginning.

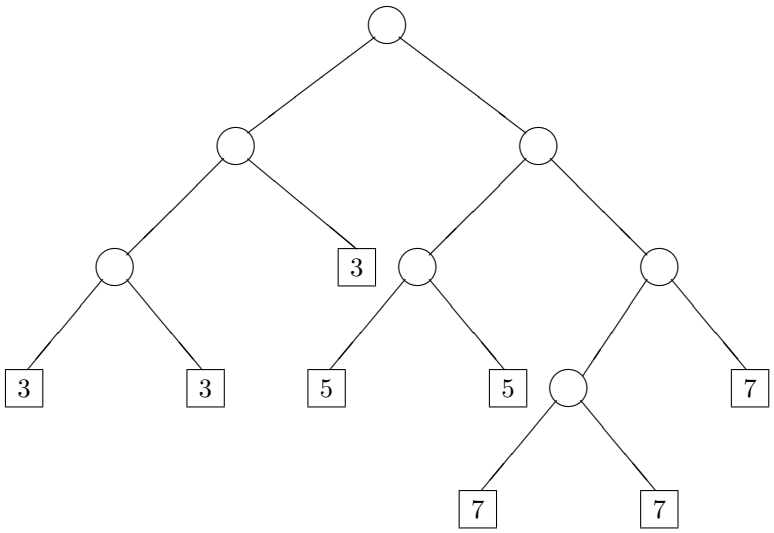


Figure 1. A 7-node binary tree.

```
Procedure GenP-Seq ( P : Pseq ; k, l, q : Integer ) ;
Begin
    If ( k < n ) Then
        p_{n-k} := p_{n-k} - 1 ;
    WritePseq ( P ) ;
    If ( k > 1 ) Then Begin
        GenP-Seq ( P, k - 1, 1, l ) ;
        If ( l < k ) And ( l < q ) Then
            GenP-Seq ( P, k, l + 1, q ) ;
    End ;
End ;
```

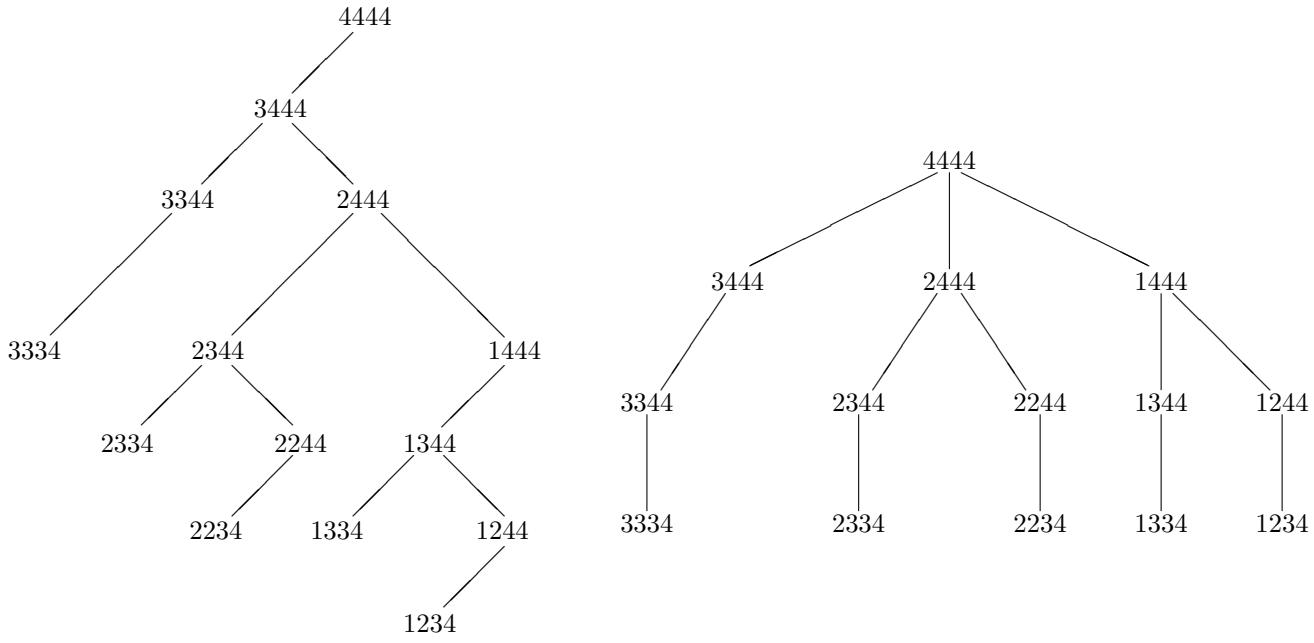
Figure 2. P-sequences generation algorithm in the reverse order of B-order.

The construction process for $n = 4$ is demonstrated in Figure 3.a by a recursion binary tree $\mathbf{T_n}$. This recursion tree is a binary tree. As we can see, the decrementation of the adjacent element of the previous decremented element in a sequence causes moving to the left child of the recursion tree and decrementation of the same element causes moving to the right child.

Now, the verification and the analysis of the algorithm are discussed. Obviously, the number of nodes in the recursion tree is equal to the number of times that *GenP-Seq* is recalled, and in each recall the next sequence is generated. With regard to the generation algorithm *GenP-Seq*, the number of times that *GenP-Seq* is recalled can be obtained from the following recurrence formula:

$$GP(k,\ l,\ q) \ = \begin{cases} 1 & \text{if } \ k = 1, \\ 1 + GP(k-1,\ 1,\ l) + GP(k,\ l+1,\ q) & \text{if } \ k > l \ \textit{and} \ q > l, \\ 1 + GP(k-1,\ 1,\ l) & \text{otherwise.} \end{cases}$$

Where $GP(k,\ l,\ q)$ denotes the number of times that algorithm *GenP-Seq* is recalled with the parameters $'P'$, $'k'$, $'l'$ and $'q'$. Here, 1 stands for the unique generated code in each recalling of the algorithm *GenP-Seq*. Since the first time the algorithm is called with $k = n$, $l = n - 1$ and $q = 1$, therefore we can write: $GP(k,\ l,\ 1) = 1 + GP(k-1,\ 1,\ l)$.



(a) (b)

Figure 3. For $n = 4$, **a)** the recursion binary tree $\mathbf{T_n}$, **b)** the equivalent transformed recursion tree $\mathbf{T'_n}$.

Lemma 1. For $k \geq 1$ and $l \leq k$,

$$GP(k,\ l,\ 1) = 1 + \sum_{\substack{j\ =\ 1 \\ j\ <\ k}}^{l} GP(k-1,\ j,\ 1).$$

Since the third parameter in $GP(k,\ l,\ 1)$ is a constant value, therefore we denote it with G_l^k. So we can write:

$$G_l^k = 1 + \sum_{\substack{j\ =\ 1 \\ j\ <\ k}}^{l} G_j^{k-1},$$

where G_1^1 is equal to 1. Now, if we show that G_{n-1}^n is equal to C_n the verification of the algorithm is proven.

Theorem. The total number of codes generated by *GenP-Seq*$(P,\ n,\ n-1,\ 1)$ is equal to C_n.

Proof. By the definition of G, we can write: $G_l^k = G_{l-1}^k + G_l^{k-1}$, where $G_0^k = 1$. It can be easily proved

$$G_l^k = \binom{k+l}{l} \frac{k-l+1}{k+1} \ .$$

Clearly

$$G_{n-1}^n = C_n = \frac{1}{n+1}\binom{2n}{n} \quad . \quad \square$$

The time required for the generation algorithm can be also obtained by the previous theorem . In order to generate C_n sequences, the algorithm is repeated C_n times, and each time one code is generated. Therefore the algorithm generates each sequence in constant average time $O(1)$.

The algorithm needs stack space to implement recursion. Since the ordering of the generation is according to the preorder traversal of the tree $\mathbf{T_n}$, therefore this space is equal to the depth of $\mathbf{T_n}$. For any n, the depth of this tree is equal to n, hence this algorithm requires a stack space of $O(n)$.

3. Ranking and Unranking Algorithms

Rank of a binary tree with respect to some ordering is the number of binary trees that come before it in the ordering. Ranking algorithms which return a unique integer from the interval $[1, C_n]$ related to a binary tree are used as a tool for data compression. An unranking algorithm determines the binary tree having a particular rank. Unranking algorithms are often used as a method of generating a random binary tree: a random integer $r \in [1, C_n]$ is extracted in a uniform way and the operation of unranking is performed [Zaks, 1980]. To represent a binary tree as an integer, we need to know its index with respect to the generation scheme of the procedure *GenP-Seq*. This is achieved by the ranking algorithm. The ordering of the generation is according to the preorder traversal of the recursion binary tree $\mathbf{T_n}$ (for $n = 4$ is denoted by Figure 3.a). This recursion binary tree can be transformed to an equivalent recursion tree [Knuth, 1973]. The equivalent recursion tree $\mathbf{T'_n}$ to the recursion binary tree $\mathbf{T_n}$ for $n = 4$ is illustrated in Figure 3.b. Clearly, preorder traversal of the recursion binary tree is the same as the depth-first search of the equivalent recursion tree, hence moving to the left children of the recursion tree $\mathbf{T_n}$

is equivalent to moving down on the levels of recursion tree $\mathbf{T'_n}$, and moving to the right children is equal to move on the adjacent subtrees.

As it is mentioned earlier, G^n_{n-1} denotes the number of nodes in the recursion binary tree $\mathbf{T_n}$. Since the recursion binary tree $\mathbf{T_n}$ is equivalent to the transformed recursion tree $\mathbf{T'_n}$, therefore G^n_{n-1} also denotes the number of nodes in the recursion tree $\mathbf{T'_n}$. From the previous equations we have:

$$G^n_{n-1} = 1 + G^{n-1}_1 + G^{n-1}_2 + \cdots + G^{n-1}_{n-1},$$

where 1 counts the root of the recursion tree $\mathbf{T'_n}$ and G^{n-1}_j counts the number of nodes in the jth subtree of the recursion tree $\mathbf{T'_n}$. In order to compute the rank of a tree, we count the number of generated trees before this tree. Let S^k_l be the number of all P-sequences of length k, beginning with $k-l+1$. Therefore we define:

$$S^k_l = \begin{cases} 0 & \text{if } \ l > k, \\ 1 & \text{if } \ l = 1, \\ G^{k-1}_{l-1} & \text{if } \ l \leq k \text{ and } k > 1\ . \end{cases}$$

Lemma 2. For $k > 1$ and $l \leq k$,

$$S^k_l = \sum_{\substack{j=1 \\ j<k}}^{l} S^{k-1}_j.$$

Clearly $C_n = G^n_{n-1} = S^{n+1}_n = \sum_{j=1}^{n} S^n_j$. By definition of G^k_l and by considering the recursion tree, we can easily observe that S^n_j's for $2 \leq j \leq n$ count the number of nodes in the $(j-1)$th subtree of the recursion tree, and S^n_1 denotes the root of recursion tree. This relation can be expanded recursively for all the subtrees.

Now, for computing the rank of a tree, by utilizing the above results, it is enough to specify the position of its corresponding code in the recursion tree. The position of a code is equal to the position of the corresponding node in the recursion tree. In order to specify the position of the following code $P_T = (p_1, p_2, \cdots, p_n)$, the difference sequence, $(m_1, m_2, \cdots, m_{n-1})$, where $m_i = n - p_i$ $(1 \leq i \leq n-1)$ is computed. The code appears in m_1th subtree of the recursion tree and $\sum_{j=1}^{m_1} S^n_j$ shows the number of generated codes in the previous subtrees including the root of the tree. If we consider the m_1th subtree as

an independent tree, then m_2 will show that the tree code has appeared in m_2th subtree of this tree, and recursively $\sum_{j=1}^{m_2} S_j^{n-1}$ will show the number of generated codes before this subtree and so on. Consequently we can write:

$$r = 1 + \sum_{i=1}^{n-1} \sum_{j=1}^{m_i} S_j^{n-i+1}.$$

Using Lemma 2, we have $\sum_{j=1}^{m_i} S_j^{n-i+1} = S_{m_i}^{n-i+2}$, and we can write:

$$r = 1 + \sum_{i=1}^{n-1} S_{m_i}^{n-i+2}.$$

Therefore the algorithm illustrated in Figure 4 computes the rank of a tree sequence in time complexity $O(n)$. It is assumed that the constants S_l^k's $(1 \leq k, l \leq n)$ in the above formula are computed in advance and stored in a two dimensional array $S[1..n, 1..n]$.

The unranking algorithm given in Figure 5 takes r in the range of $1 \cdots C_n$ as input and returns the P-sequence $P_T = (p_1, p_2, \cdots, p_n)$ of the corresponding binary tree. The unranking algorithm essentially reverses the steps carried out in computing the rank. According to the rank of a tree, the position of its sequence in the recurrence tree, is specified. The position of a sequence in the recurrence tree, depends on the number of times that each element is decremented. Therefore, this position is obtained by using S_l^k's $(1 \leq k, l \leq n+1)$. In the unranking algorithm, the value of all p_i's $(i = 1, \cdots, n)$ are initially assigned to n. In the next step, the maximum j for which $S_j^{n+1} < r$ is found. Then p_1 is decremented by j, and later $r = r - S_j^{n+1}$ is assigned. For evaluating p_2,

```
Function Rank ( P : Pseq ) : Integer ;
Var r, i : Integer ;
Begin
    r := 0 ;
    For i := 1 To n − 1 Do
        r := r + S_{n−p_i}^{n−i+2} ;
    Rank := r + 1 ;
End ;
```

Figure 4. Rank algorithm.

```
Function Unrank ( r : Integer ) : Pseq ;
Var P : Pseq ;  i, j : Integer ;
Begin
    For i := 1 To n Do
        p_i := n ;
    i := n + 1 ; j := n − 1 ;
    While ( j <> 0 ) Do
        If ( S_j^i < r ) Then Begin
            r := r − S_j^i ;
            i := i − 1 ;
            p_{n−i+1} := p_{n−i+1} − j ;
        End
        Else
            j := j − 1 ;
    Unrank:= P;
End ;
```

Figure 5. Unrank algorithm.

next maximum j is computed such that $S_j^n < r$, and p_2 is decremented by j, then we set $r = r - S_j^n$. The above operations are repeated till in computing any p_i, we can not find a j such that $S_j^{n+2-i} < r$. Considering the above discussion, the complexity of the unranking algorithm is $O(n)$.

4. Construction Algorithm

In this section a construction algorithm for a binary tree from a P-sequence is described. The algorithm *MakeTreeP* illustrated in Figure 6 takes a P-sequence as an input and its output is a binary tree which is constructed in a linked structure. The algorithm performs as follows. For a given P-sequence $P = (p_1, p_2, \cdots, p_n)$, initially a dummy root is created and later *MakeTreeP* is called with three parameters: $'Tree'$, $'True'$ and $'p_1'$, where $'Tree'$ shows the address of dummy root. For this dummy root one right child is created, then $p_1 - 1$ new nodes are created such that each of them is a left child of the

```
Procedure MakeTreeP ( Tree : TreePtr ; Sw : Boolean ; j : Integer ) ;
Begin
    If ( Sw = True ) Then Begin
        MakeNode ( Tree↑ . Right ) ;
        Tree := Tree↑ . Right ;
    End
    Else Begin
        MakeNode ( Tree↑ . Left ) ;
        Tree := Tree↑ . Left ;
    End ;
    If ( j − 1 > 0 ) Then
        MakeTreeP ( Tree, False, j − 1 ) ;
    { i is a global variable and initially set to 1 }
    i := i + 1 ;
    If ( i <= n ) Then
        If ( p_i − p_{i−1} > 0 ) Then
            MakeTreeP ( Tree, True, p_i − p_{i−1} ) ;
End ;
```

Figure 6. Construction algorithm of a binary tree from a P-sequence.

previous created node, simultaneously the address of all the created nodes are pushed into a stack (recursively). At this stage the current node is the last created node. Now for all $2 \leq i \leq n$, if $p_i - p_{i-1} \neq 0$, then one new node as a right child of the current node is created. Later $p_i - p_{i-1} - 1$ new nodes as left children, are constructed such that each of them is a left child of the previous created node, and their corresponding address are pushed into the stack. These operations are controlled by the variable $'Sw'$, where $'Sw'$ is the second parameter in the algorithm and initially is set to $True$. If $p_i - p_{i-1} = 0$, then an address is popped out of the stack and the process is continued from the beginning.

It is well known that the construction algorithm establishes a $1 - 1$ correspondence between the sequences and the set of binary trees of order n.

5. Conclusion

A new algorithm for the generation of the P-sequences is presented. The algorithm generates each sequence in constant average time $O(1)$. The time complexity of ranking and unranking algorithms presented for both sequences is $O(n)$.

It should be noted that, the generation algorithm can be modified such that to generate the sequences in different order. By changing the position of elements in the decrement instruction inside the algorithm, it is possible to generate P-sequences in Ballot-order. The average time complexity of the new generation algorithm is also $O(1)$. Clearly, their corresponding ranking and unranking algorithms with new orders can be easily written in $O(n)$.

Reference

Ahrabian, H., and Nowzari-Dalini, A., (1999) On the generation of binary trees in A-order. *International Journal of Computer Mathematics*, vol.71, pp.1-7.

Bultena, B., and Ruskey, F., (1998) An Eades-McKay algorithm for well-formed parentheses string. *Information Processing Letters*, vol.68, pp.255-259.

Gupta, D.K., (1991) On the generation of P-sequences. *International Journal of Computer Mathematics*, vol.38, pp.31-35.

Knuth, D.E., (1973) *The Art of Computer Programming, Vol. 1: Fundemental Algorithms.* (Second Edition), Addison-Wesley, Massachusetts.

Pallo, J., and Racca, R., (1985) A note on generating binary tree in A-order and B-order. *International Journal of Computer Mathematics*, vol.18, pp.27-39.

Rotem, D., (1975) On a correspondence between binary tree and certain type of permutation. *Information Processing Letters*, vol.4, pp.58-61.

Rotem, D., and Varol, Y.L., (1978) Generation of binary trees from Ballot-sequences. *Journal of the ACM*, vol.25, pp.396-404.

Ruskey, F., and Hu, T.C., (1977) Generating binary tree lexicographically. *SIAM Journal on Computing*, vol.6, pp.745-758.

Vajnovszki, V., (1998) On the loopless generation of binary tree sequences. *Information Processing Letters*, vol.68, pp.113-117.

Zaks, S., (1980) Lexicographic generation of ordered tree. *Theoretical Computer Science*, vol.10, pp.63-82.